

AN EFFICIENT IMPLEMENTATION OF BREZZI-DOUGLAS-MARINI (BDM) MIXED FINITE ELEMENT METHOD IN MATLAB

SHUN ZHANG

Abstract. In this paper, a MATLAB package `bdm_mfem` for a linear Brezzi-Douglas-Marini (BDM) mixed finite element method is provided for the numerical solution of elliptic diffusion problems with mixed boundary conditions on unstructured grids. BDM basis functions defined by standard barycentric coordinates are used in the paper. Local and global edge ordering are treated carefully. MATLAB build-in functions and vectorizations are used to guarantee the correctness of the programs. The package is simple and efficient, and can be easily adapted for more complicated edge-based finite element spaces. A numerical example is provided to illustrate the usage of the package.

1. INTRODUCTION

In recent years, MATLAB is widely used in the numerical simulation and is proved to be an excellent tool for academic educations. For example, Trefethen's book on spectral methods [15] is extremely popular. In the area of finite element method, there are several papers on writing clear, short, and easily adapted MATLAB codes, for example [1, 2, 10, 11]. Vectorizations are used in [10] and [11] to guarantee the effectiveness of the MATLAB finite elements codes. The mixed finite element [13, 3, 4] is now widely used in many area of scientific computation. For example, in [5, 6, 7, 8, 9], we use RT(Raviart-Thomas)/BDM(Brezzi-Douglas-Marini) space to build recovery-based a posteriori error estimators. On the other side, except for the clear presentation of [2] on RT_0 , the implementation of more complicated BDM elements is still somehow confusing for researchers and students. The purpose of this paper is to fill this gap by giving a simple, efficient, and easily adaptable MATLAB implementation of BDM_1-P_0 mixed finite element methods for of elliptic diffusion problems with mixed boundary conditions on unstructured grids.

For linear BDM elements, there are several versions to write the basis functions explicitly. In [4, 14], basis functions are defined on each elements using heights and normal vectors. This version of basis functions is less straightforward than the basis functions defined by barycentric coordinates. After all, everyone is very familiar with barycentric coordinates in finite element programms. Thus, in this implementation, we will use the definition which only uses barycentric coordinates.

There are two basis functions on each edge for linear BDM elements. Unlike the RT element, BDM basis functions depend on the stating and terminal points of the edge. Thus, when assembling the local matrix, for a local edge on each element, we need to make sure we find its correct global stating and terminal points of the edge. There is an implementation

Date: August 27, 2015.

Key words and phrases. MATLAB; mixed finite element method; Brezzi-Douglas-Marini element; Raviart-Thomas element; BDM element; RT element.

This work was supported in part by Research Grants Council of the Hong Kong SAR, China under the GRF Grant Project No. 11303914, CityU 9042090.

of BDM element in *iFEM* package [10] <https://bitbucket.org/ifem/ifem/>. But in order to make the local ordering of edges in an element is the same as the global ordering of edges, the triangles are not always counterclockwisely oriented. This will cause confusion for programmers. And if we use this ordering, sometime we may need two kinds of element map, one is counterclockwisely ordered, the other is ordered by the indices of vertices. This will make things more complicated. In this implementation, we will use the standard counterclockwisely ordering of vertices of triangles.

Besides these issues, to get the right convergence order, we need to handle the boundary conditions carefully. In this package, we use the basic data structure of *iFEM* [10], and full MATLAB vectorization is used.

In a summery, in this package:

- (1) BDM_1 basis functions are explicitly defined using barycentric coordinates.
- (2) Elements are still positively oriented. A function is used to find the right global stating and terminal vertices of an edge in a local element.
- (3) Mixed type of boundary conditions are handled correctly to guarantee the right order of convergence.
- (4) MATLAB build-in functions and vectorizations are used to guarantee the erectness of the programs.

The package can be download from http://personal.cityu.edu.hk/~szzhang26/bdm_ifem.zip.

The paper is organized as follows. Section 2 describes the model diffusion problem, its BDM_1-P_0 mixed finite element approximation and the corresponding matrix problem. Section 3 introduces a simple example problem to demonstrate our MATLAB code. In Section 4, edge-based linear BDM basis functions are defined. The main part of the code for the matrix problem is discussed in Section 6. MATLAB functions to checking the errors are discussed in Section 7. Finally, we discuss some related finite elements in Section 8.

2. MODEL PROBLEM AND BDM_1-P_0 MIXED FINITE ELEMENT METHOD

2.1. Model problem. Let Ω be a bounded polygonal domain in \mathbb{R}^2 , with boundary $\partial\Omega = \bar{\Gamma}_D \cup \bar{\Gamma}_N$, $\Gamma_D \cap \Gamma_N = \emptyset$, and measure(Γ_D) $\neq 0$, and let \mathbf{n} be the outward unit vector normal to the boundary. For a vector function $\boldsymbol{\tau} = (\tau_1, \tau_2)$, define the divergence and curl operators by $\nabla \cdot \boldsymbol{\tau} = \frac{\partial \tau_1}{\partial x_1} + \frac{\partial \tau_2}{\partial x_2}$ and $\nabla \times \boldsymbol{\tau} = \frac{\partial \tau_2}{\partial x_1} - \frac{\partial \tau_1}{\partial x_2}$. For a function v , define the gradient and rotation operators by $\nabla v = (\frac{\partial v}{\partial x_1}, \frac{\partial v}{\partial x_2})^t$ and $\nabla^\perp v = (\frac{\partial v}{\partial x_2}, -\frac{\partial v}{\partial x_1})^t$.

Consider diffusion equation

$$(2.1) \quad -\nabla \cdot (\alpha(x)\nabla u) = f \quad \text{in } \Omega$$

with boundary conditions

$$(2.2) \quad -\alpha \nabla u \cdot \mathbf{n} = g_N \quad \text{on } \Gamma_N \quad \text{and} \quad u = g_D \quad \text{on } \Gamma_D.$$

We assume that the right-hand side $f \in L^2(\Omega)$, g_D in $H^{1/2}(\Gamma_D)$, and that $g_N \in L^2(\Gamma_N)$, and that $\alpha(x)$ is a positive piecewise constant function. Define the flux by $\boldsymbol{\sigma} = -\alpha(x)\nabla u$, then we have

$$(2.3) \quad \alpha^{-1}\boldsymbol{\sigma} = -\nabla u \quad \text{and} \quad \nabla \cdot \boldsymbol{\sigma} = f.$$

Define the standard $H(\text{div}; \Omega)$ spaces as

$$\begin{aligned} H(\text{div}; \Omega) &:= \{\boldsymbol{\tau} \in L^2(\Omega)^2 : \nabla \cdot \boldsymbol{\tau} \in L^2(\Omega)\}, \\ H_N(\text{div}; \Omega) &:= \{\boldsymbol{\tau} \in H(\text{div}; \Omega) : \boldsymbol{\tau} \cdot \mathbf{n} = 0 \text{ on } \Gamma_N\}. \end{aligned}$$

Multiply the first equation in (2.3) by a $\boldsymbol{\tau} \in H_N(\text{div}; \Omega)$ and integrating by parts, we get

$$(\alpha^{-1}\boldsymbol{\sigma}, \boldsymbol{\tau}) = -(\nabla u, \boldsymbol{\tau}) = (\nabla \cdot \boldsymbol{\tau}, v) - (\boldsymbol{\tau} \cdot \mathbf{n}, g_D)_{\Gamma_D}$$

where $(\cdot, \cdot)_\omega$ is the L^2 inner product on a domain ω . If $\omega = \Omega$, we omit the subscript. Then the mixed variational formulation is to find $(\boldsymbol{\sigma}, u) \in H(\text{div}; \Omega) \times L^2(\Omega)$ with $\boldsymbol{\sigma} \cdot \mathbf{n} = g_N$ on Γ_N , such that

$$(2.4) \quad \begin{cases} (\alpha^{-1}\boldsymbol{\sigma}, \boldsymbol{\tau}) - (\nabla \cdot \boldsymbol{\tau}, u) &= -(\boldsymbol{\tau} \cdot \mathbf{n}, g_D)_{\Gamma_D} \quad \forall \boldsymbol{\tau} \in H_N(\text{div}; \Omega), \\ (\nabla \cdot \boldsymbol{\sigma}, v) &= (f, v) \quad \forall v \in L^2(\Omega). \end{cases}$$

The existence, uniqueness, and stability results of (2.4) are well-known, and can be found in standard references, for example, [4].

2.2. BDM_1-P_0 mixed formulation. Let $\mathcal{T} = \{K\}$ be a regular triangulation of the domain Ω . Denote the set of all edges of the triangulation by $\mathcal{E} := \mathcal{E}_I \cup \mathcal{E}_D \cup \mathcal{E}_N$, where \mathcal{E}_I is the set of all interior element edges and \mathcal{E}_D and \mathcal{E}_N are the sets of all boundary edges belonging to the respective Γ_D and Γ_N . For any element $K \in \mathcal{T}$, denote by $P_k(K)$ the space of polynomials on K with total degree less than or equal to k . The $H(\text{div}; \Omega)$ conforming Brezzi-Douglas-Marini (BDM) space [3] of the lowest order is defined by

$$BDM_1 = \{\boldsymbol{\tau} : \boldsymbol{\tau}|_K \in BDM_1(K), \forall K \in \mathcal{T}\} \quad \text{with} \quad BDM_1(K) = P_1(K)^2.$$

and let $BDM_{1,N} = BDM_1 \cap H_N(\text{div}; \Omega)$. The piecewise constant space P_0 is defined by

$$P_0 = \{v : v|_K \in P_0(K), \forall K \in \mathcal{T}\}.$$

For simplicity, we further assume that α is a positive constant in each element $K \in \mathcal{T}$. Let \mathcal{T}_N be the one dimensional mesh induced by \mathcal{T} on Γ_N . Define

$$P_1(\mathcal{T}_N) = \{v : v|_E \in P_1(E), \forall E \in \mathcal{T}_N\}.$$

Let $g_{N,h}$ be the L^2 projection of g_N on $P_1(\mathcal{T}_N)$. The BDM_1-P_0 mixed finite element discrete problem then is to find $(\boldsymbol{\sigma}_h, u_h) \in BDM_1 \times P_0$ with $\boldsymbol{\sigma}_h \cdot \mathbf{n} = g_{N,h}$ on Γ_N , such that

$$(2.5) \quad \begin{cases} (\alpha^{-1}\boldsymbol{\sigma}_h, \boldsymbol{\tau}_h) - (\nabla \cdot \boldsymbol{\tau}_h, u_h) &= -(\boldsymbol{\tau}_h \cdot \mathbf{n}, g_D)_{\Gamma_D} \quad \forall \boldsymbol{\tau}_h \in BDM_{1,N}, \\ -(\nabla \cdot \boldsymbol{\sigma}_h, v_h) &= -(f, v_h) \quad \forall v_h \in P_0. \end{cases}$$

The discrete problem (2.5) has a unique solution, and the following error estimates hold assuming that the solution $(\boldsymbol{\sigma}, u)$ has enough regularity:

$$(2.6) \quad \|\alpha^{-1/2}(\boldsymbol{\sigma} - \boldsymbol{\sigma}_h)\|_0 \leq Ch^2\|\alpha^{-1/2}\boldsymbol{\sigma}\|_2 \quad \text{and} \quad \|u - u_h\|_0 \leq C(h\|u\|_1 + h^2\|\boldsymbol{\sigma}\|_2),$$

where h is the mesh size and $\|\cdot\|_k$ is the standard norm of Sobolev space H^k . Thus, we should expect order 2 convergence of $\boldsymbol{\sigma}$ and order 1 convergence of u if the solution is smooth enough.

Let $\boldsymbol{\sigma}_N \in BDM_1$ be the interpolation of $g_{N,h}$ on Γ_N (The construction of $\boldsymbol{\sigma}_N$ will be explained in Section ?? in detail). Then $\boldsymbol{\sigma}_h = \boldsymbol{\sigma}_N + \boldsymbol{\sigma}_0$, with $\boldsymbol{\sigma}_0 \in BDM_{1,N}$ solves the following discrete problem:

$$(2.7) \quad \begin{cases} (\alpha^{-1}\boldsymbol{\sigma}_0, \boldsymbol{\tau}_h) - (\nabla \cdot \boldsymbol{\tau}_h, u_h) &= -(\boldsymbol{\tau}_h \cdot \mathbf{n}, g_D)_{\Gamma_D} - (\alpha^{-1}\boldsymbol{\sigma}_N, \boldsymbol{\tau}_h) \quad \forall \boldsymbol{\tau}_h \in BDM_{1,N}, \\ -(\nabla \cdot \boldsymbol{\sigma}_0, v_h) &= -(f, v_h) + (\nabla \cdot \boldsymbol{\sigma}_N, v_h) \quad \forall v_h \in P_0. \end{cases}$$

2.3. Matrix problem. Suppose that all edges are uniquely defined with fixed initial and terminal vertices, in Section 5, we will define two basis functions $\phi_{j,1}$ and $\phi_{j,2}$ for an edge $E_j \in \mathcal{E}$, $j = 1, \dots, NE$, with NE is the number of edges. For simplicity, we assume that the total number of all edges in \mathcal{E}_I and \mathcal{E}_D is M , and $BDM_{1,N} = \text{span}\{\phi_{j,1}, \phi_{j,2}, j = 1 \dots, M\}$ (The real mesh may has a different order of indices). The basis of P_0 is very simple. For the element K_j , its basis is 1_j which is 1 on K_j and 0 elsewhere.

We want to compute the coefficient vectors $\mathbf{x} \in \mathbb{R}^{2NE}$ of $\boldsymbol{\sigma}_h$ and $\mathbf{y} \in \mathbb{R}^{NT}$ of u_h with respect to the BDM_1 basis and P_0 basis

$$(2.8) \quad \boldsymbol{\sigma}_h = \sum_{j=1}^{NE} (x_j \phi_{j,1} + x_{NE+j} \phi_{j,2}) \quad \text{and} \quad u_h = \sum_{k=1}^{NT} y_k 1_k,$$

and

$$(2.9) \quad \boldsymbol{\sigma}_N = \sum_{j=M+1}^{NE} (x_j \phi_{j,1} + x_{NE+j} \phi_{j,2})$$

with x_j and x_{NE+j} are determined by $g_{N,h}$ (discussed later in Section ??). Then

$$\begin{aligned} & \sum_{j=1}^M (x_j (\alpha^{-1} \phi_{i,1}, \phi_{j,1}) + x_{j+NE} (\alpha^{-1} \phi_{i,1}, \phi_{j,2})) - \sum_{k=1}^{NT} y_k (\phi_{i,1}, 1)_{K_k} \\ &= -(g_D, \phi_{i,1} \cdot \mathbf{n}_\Omega)_{\Gamma_D} - \sum_{j=M+1}^{NE} (x_j (\alpha^{-1} \phi_{i,1}, \phi_{j,1}) + x_{j+NE} (\alpha^{-1} \phi_{i,1}, \phi_{j,2})); \\ & \sum_{j=1}^M (x_j (\alpha^{-1} \phi_{i,2}, \phi_{j,1}) + x_{j+NE} (\alpha^{-1} \phi_{i,2}, \phi_{j,2})) - \sum_{k=1}^{NT} y_k (\phi_{i,2}, 1)_{K_k} \\ &= -(g_D, \phi_{i,2} \cdot \mathbf{n}_\Omega)_{\Gamma_D} - \sum_{j=M+1}^{NE} (x_j (\alpha^{-1} \phi_{i,2}, \phi_{j,1}) + x_{j+NE} (\alpha^{-1} \phi_{i,2}, \phi_{j,2})); \\ & - \sum_{j=1}^M (x_j (\phi_{j,1}, 1)_{K_\ell} + x_{j+NE} (\phi_{j,2}, 1)_{K_\ell}) \\ &= -(f, 1)_{K_\ell} + \sum_{j=M+1}^{NE} (x_j (\nabla \cdot \phi_{j,1}, 1)_{K_\ell} + x_{j+NE} (\nabla \cdot \phi_{j,2}, 1)_{K_\ell}); \end{aligned}$$

for $i = 1, \dots, M$, $\ell = 1, \dots, NT$. The coefficients x_j , x_{j+NE} , $j = M + 1, \dots, NE$ are known from $g_{N,h}$. Rewrite it as a matrix problem, we have

$$(2.10) \quad \begin{pmatrix} B & C^t \\ C & 0 \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \begin{pmatrix} b1 \\ b2 \end{pmatrix},$$

where B is a $2M \times 2M$ matrix and C is a $NT \times 2M$ matrix with entries:

$$(2.11) \quad \begin{aligned} B_{i,j} &= (\alpha^{-1}\phi_{i,1}, \phi_{j,1}), & B_{i,j+M} &= (\alpha^{-1}\phi_{i,1}, \phi_{j,2}), & B_{i+M,j} &= (\alpha^{-1}\phi_{i,2}, \phi_{j,1}), \\ B_{i+M,j+M} &= (\alpha^{-1}\phi_{i,2}, \phi_{j,2}), & C_{\ell,j} &= -(\nabla \cdot \phi_{j,1}, 1_\ell), & C_{\ell,j+M} &= -(\nabla \cdot \phi_{j,2}, 1_\ell), \end{aligned}$$

where $i = 1, \dots, M$, $j = 1, \dots, M$, and $\ell = 1, \dots, NT$; and the right hand side where $b1$ is a $2M \times 1$ vectors and $b2$ is an $NT \times 1$ vector with entries

$$\begin{aligned} b1_i &= -(g_D, \phi_{i,1} \cdot \mathbf{n}_\Omega)_{\Gamma_D} - \sum_{j=M+1}^{NE} (x_j(\alpha^{-1}\phi_{i,1}, \phi_{j,1}) + x_{j+NE}(\alpha^{-1}\phi_{i,1}, \phi_{j,2})); \\ b1_{i+M} &= -(g_D, \phi_{i,2} \cdot \mathbf{n}_\Omega)_{\Gamma_D} - \sum_{j=M+1}^{NE} (x_j(\alpha^{-1}\phi_{i,2}, \phi_{j,1}) + x_{j+NE}(\alpha^{-1}\phi_{i,2}, \phi_{j,2})); \\ b2_\ell &= -(f, 1)_{K_\ell} + \sum_{j=M+1}^{NE} (x_j(\nabla \cdot \phi_{j,1}, 1)_{K_\ell} + x_{j+NE}(\nabla \cdot \phi_{j,2}, 1)_{K_\ell}) \end{aligned}$$

where $i = 1, \dots, M$ and $\ell = 1, \dots, NT$; $(x_1, \dots, x_M, x_{1+NE}, \dots, x_{M+NE}, y_1, \dots, y_{NT})^t$ is the $2M + NT \times 1$ solution vector.

3. A NUMERICAL EXAMPLE

We will demonstrate our MATLAB program by a simple test problem. Let $\Omega = (-1, 1)^2$, with $\Gamma_N = \{x \in (-1, 1)\} \times \{y = 1\}$, and the rest is Γ_D . The mesh is given in Fig. 3. We choose the diffusion coefficient and the exact solution to be

$$\alpha = \begin{cases} 10 & \text{if } x < 0; \\ 1 & \text{if } x > 0; \end{cases} \quad \text{and} \quad u(x, y) = \begin{cases} (x^2 y^2 + x)/10 + y & \text{if } x < 0; \\ x^2 y^2 + x + y & \text{if } x > 0. \end{cases}$$

The right-hand side $f = -2(x^2 + y^2)$. The exact σ is

$$\sigma(x, y) = \begin{cases} -(2xy^2 + 1, 2x^2y + 10)^t & \text{if } x < 0; \\ -(2xy^2 + 1, 2x^2y + 1)^t & \text{if } x > 0. \end{cases}$$

It's clear that σ itself is not continuous, but its normal component is continuous. The boundary conditions are

$$u(x, y) = \begin{cases} y^2/10 + y - 1/10 & \text{if } x = -1; \\ y^2 + y + 1 & \text{if } x = 1; \\ (x^2 + x)/10 - 1 & \text{if } x < 0 \text{ and } y = -1; \\ x^2 + x - 1 & \text{if } x > 0 \text{ and } y = -1; \end{cases}$$

and

$$-\alpha \nabla u \cdot (0, 1)^t = g_N = \begin{cases} -2x^2 - 10 & \text{if } x < 0 \text{ and } y = 1; \\ -2x^2 - 1 & \text{if } x > 0 \text{ and } y = 1. \end{cases}$$

We choose this exact solution to emphasize that the flux σ is in $H(\text{div}; \Omega)$, but not the ∇u . The main MATLAB codes of α , f , g_D and g_N are given in `exactalpha.m`, `f.m`,

LISTING 1. exactalpha, f, gD, and gN

```

1 function z = exactalpha(p)
2 ix = (p(:,1)<0); z = ones(size(p,1),1); z(ix) = 10.0;
3 end
4 function z = f(p)
5 x = p(:,1); y = p(:,2); z = -2*(x.*x+y.*y);
6 end
7 function z = gD(p)
8 x = p(:,1); y = p(:,2);
9 z=(x<0).* (0.1*x.*x.*y.*y+0.1*x*y)+(x>=0).* (x.*x.*y.*y+x*y);
10 end
11 function z = gN(p)
12 x = p(:,1); y = p(:,2); z=(x<0).* (-2*x.*x-10)+(x>=0).* (-2*x.*x-1);
13 end

```

`gD.m`, and `gN.m` respectively. Here the Dirichlet boundary condition is given by the exact solution in `gD.m` for simplicity.

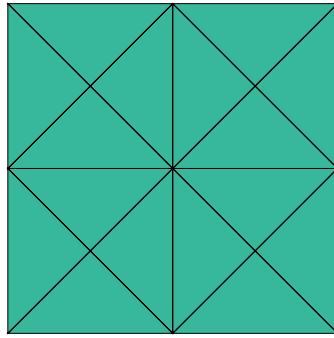


FIGURE 1. mesh of the example

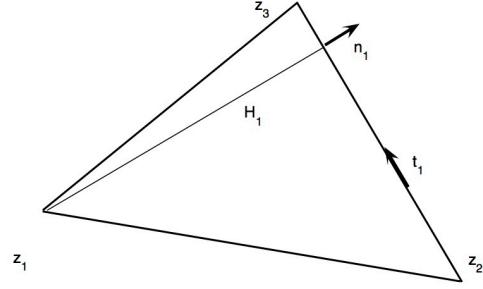


FIGURE 2. a triangle

The MATLAB program to solve this problem is given in `main.m`. Lines 2-7 read the essential geometric data of the problem. Line 9 generates the edges, and other necessary geometric relations. Lines 11-12 solve the problem by the BDM_1 mixed finite element method.

4. TRIANGULATION AND GEOMETRIC DATA STRUCTURES

4.1. Geometric description and geometric relations. We follow [10] for the data representation of the set of all vertices, the regular triangulation \mathcal{T} , the edges, and the boundaries.

The set of all vertices $\mathcal{N} = \{z_1, \dots, z_N\}$ is represented by an $N \times 2$ matrix `node(1:N, 1:2)`, where N is the number of vertices, and the i -th row of `node` is the coordinates of the i -th vertex $z_i = (x_i, y_i)$, `node(i, :) = [xi, yi]`. Lines 2-3 of `main.m` give the node of our example. For example, the 1st vertex has coordinates $x_1 = -1$ and $y_1 = 1$.

LISTING 2. main.m

```

1 %% Geometry setting
2 node = [-1 1; 0 1; 1 1; -0.5 0.5; 0.5 0.5; -1 0; 0 0; 1 0; ...
3      -0.5 -0.5; 0.5 -0.5;-1.0 -1.0; 0.0 -1.0;1.0 -1.0];
4 elem =[4 2 1;4 1 6;4 6 7;4 7 2;5 3 2;5 2 7;5 7 8;5 8 3;9 7 6;...
5      9 6 11;9 11 12;9 12 7;10 8 7;10 7 12;10 12 13;10 13 8];
6 bdEdge = [2 0 0;1 0 0; 0 0 0;0 0 0;2 0 0;0 0 0;0 0 0;0 0 0;1 0 0;...
7      0 0 0;1 0 0;1 0 0;0 0 0;0 0 0;0 0 0;1 0 0;1 0 0];
8 %% Geometric relations
9 [edge,elem2edge,signedge] = geomrelations(elem);
10 %%
11 [sigma,u] = diffusionbdm(node,elem,bdEdge,elem2edge,edge, ...
12     signedge,@exactalpha,@f,@gD,@gN);

```

LISTING 3. geomrelations.m

```

1 function [edge,elem2edge,signedge] = geomrelations(elem)
2 NT = size(elem,1);
3 totalEdge = sort([elem(:,[2,3]); elem(:,[3,1]); elem(:,[1,2])],2);
4 [edge, useless, j] = unique(totalEdge, 'rows');
5 elem2edge = reshape(j,NT,3);
6
7 signedge = ones(NT,3);
8 signedge(:,1) = signedge(:,1) - 2* (elem(:,2)>elem(:,3));
9 signedge(:,2) = signedge(:,2) - 2* (elem(:,3)>elem(:,1));
10 signedge(:,3) = signedge(:,3) - 2* (elem(:,1)>elem(:,2));
11 end

```

The triangulation \mathcal{T} is represented by an $NT \times 3$ matrix $\text{elem}(1:NT, 1:3)$ with NT the number of elements. The i -th element $K_i = \text{conv}\{z_i, z_j, z_k\}$ is stored as $\text{elem}(i, :) = \dots [i \ j \ k]$, where the vertices are given in the counterclockwise order. Lines 4-5 of `main.m` give the `elem` of our example. For example, the 1st element K_1 has three vertices in the order of z_4, z_2 and z_1 .

We call the the opposite edge of the i -th vertex, $i = 1, 2, 3$ of a triangle the i -th edge of the triangle.

The matrix `bdEdge(1:NT, 1:3)` indicates which edge of an element is on the boundary of the domain. For a non-boundary edge, the value is 0; the value is 1 or 2 for a Dirichlet or Neumann boundary edge respectively. Lines 4-5 of `main.m` give the `bdEdge` of our example. For example, the 1st element K_1 has three edge, the first edge is on the Neumann boundary, and the other two are not on the boundary.

The MATLAB code `geomrelations.m` generates `edge`, `elem2edge`, and `signedge`. The explanations of these codes can be found in [10].

The matrix `edge(1:NE, 1:2)` is defined with NE the total number of edges, and the k -th edge E_k with the starting vertex z_i and the terminal vertex z_j is stored as `edge(k, :) = ... [i \ j]`. We always ensure that `edge(k, 1) < edge(k, 2)`. Lines 3-4 of `geomrelations.m`

LISTING 4. gradlambda.m

```

1 function [a,b,area] = gradlambda(node,elem)
2 n1 = elem(:,1); n2 = elem(:,2); n3 = elem(:,3);
3 NT = size(elem,1); a = zeros(NT,3); b = zeros(NT,3);
4 a(:,1) = node(n2,2)-node(n3,2); b(:,1) = node(n3,1)-node(n2,1);
5 a(:,2) = node(n3,2)-node(n1,2); b(:,2) = node(n1,1)-node(n3,1);
6 a(:,3) = node(n1,2)-node(n2,2); b(:,3) = node(n2,1)-node(n1,1);
7 area = (a(:,2).*b(:,3)-a(:,3).*b(:,2))/2.0;
8 end

```

generate edge. For our example, $\text{NE}=28$ and $\text{edge}=[1\ 2; 1\ 4; 1\ 6; 2\ 3; 2\ 4; 2\ \dots; 5; 2\ 7; 3\ 5; 3\ 8; 4\ 6; 4\ 7; 5\ 7; 5\ 8; 6\ 7; 6\ 9; 6\ 11; 7\ 8; 7\ 9; 7\ 10; \dots; 7\ 12; 8\ 10; 8\ 13; 9\ 11; 9\ 12; 10\ 12; 10\ 13; 11\ 12; 12\ 13]$. For example, the 2nd edge's starting vertex is z_1 and terminal vertex is z_4 .

For an edge E_j with the starting vertex z_s and the terminal vertex z_t , its normal direction is defined as $(y_t - y_s, x_s - x_t)^t / |E_j|$.

The matrix $\text{elem2edge}(1:\text{NT}, 1:3)$ is the matrix whose k -th row represents the 3 global indices of the edges in the order of local edges. Line 5 of `geomrelations.m` generates `elem2edge`. For our example, $\text{elem2edge}=[1\ 2\ 5; 3\ 10\ 2; 14\ 11\ 10; 7\ \dots; 5\ 11; 4\ 6\ 8; 7\ 12\ 6; 17\ 13\ 12; 9\ 8\ 13; 14\ 15\ 18; 16\ 23\ 15; 27\ 24\ 23; \dots; 20\ 18\ 24; 17\ 19\ 21; 20\ 25\ 19; 28\ 26\ 25; 22\ 21\ 26]$. For example, the 2nd element's three edges are E_3 , E_{10} , and E_2 .

Each edge has its fixed global orientation, while on each element, it has a local orientation, $\text{localEdge}=[2\ 3; 3\ 1; 1\ 2]$. The matrix `signedge` is an $\text{NT} \times 3$ matrix with value 1 denoting the local and global orientations are the same, and -1 denoting that they are different. For our example, $\text{signedge}=[-1\ 1\ -1; 1\ -1\ -1; 1\ -1\ 1; -1\ 1\ 1; -1\ \dots; 1\ -1; 1\ -1\ -1; 1\ -1\ 1; -1\ 1\ -1; 1\ -1\ -1; 1\ -1\ 1; -1\ 1\ 1; -1\ 1\ \dots; -1; 1\ -1\ -1; 1\ -1\ 1; -1\ 1\ 1]$.

4.2. Barycentric coordinate and its gradient. On an element K with counterclockwise vertices $\{z_1, z_2, z_3\}$, we define the barycentric coordinate $\lambda_i = (a_i x + b_i y + c_i) / (2|K|)$, $i = 1, 2, 3$, such that $\lambda_i(z_j) = \delta_{ij}$. We only need to compute the gradient (and curl, rot, div) operators in this paper, so we only need to compute a_i , b_i , and the area $|K|$. The formulas are

$$\nabla \lambda_i = \frac{1}{2|K|} \begin{pmatrix} y_{i+1} - y_{i+2} \\ x_{i+2} - x_{i+1} \end{pmatrix} \quad \text{and} \quad 2|K| = \det \begin{pmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{pmatrix}.$$

The function `gradlambda.m` computes the coefficients `a`, `b`, and `area`. Here `a` and `b` are two $\text{NT} \times 3$ matrices, with each row stores the coefficients a and b for the three barycentric coordinates of corresponding 3 vertices.

4.3. Normal and tangential vectors. On a local element K with counterclockwise oriented vertices $\{z_1, z_2, z_3\}$, and $E_1 = \{z_2, z_3\}$, $E_2 = \{z_3, z_1\}$, and $E_3 = \{z_1, z_2\}$ (Figure 3), we will discuss the ∇ and ∇^\perp of λ_2 as examples:

$$\nabla^\perp \lambda_2 = \frac{1}{2|K|} \begin{pmatrix} y_3 - y_1 \\ x_1 - x_3 \end{pmatrix} \quad \text{and} \quad \nabla^\perp \lambda_2 = \frac{1}{2|K|} \begin{pmatrix} x_1 - x_3 \\ y_1 - y_3 \end{pmatrix}.$$

where $|K|$ is the area of K . On edge $E_1 = \{z_2, z_3\}$, the unit tangential and normal vectors are

$$\mathbf{t}_{E_1} = \frac{1}{|E_1|} \begin{pmatrix} x_3 - x_2 \\ y_3 - y_2 \end{pmatrix} \quad \text{and} \quad \mathbf{n}_{E_1} = \frac{1}{|E_1|} \begin{pmatrix} y_3 - y_2 \\ x_2 - x_3 \end{pmatrix}.$$

where $|E|$ is the length of E . From Figure 3, we have $\begin{pmatrix} x_1 - x_3 \\ y_1 - y_3 \end{pmatrix} \cdot \mathbf{n}_{E_1} = \begin{pmatrix} y_3 - y_1 \\ x_1 - x_3 \end{pmatrix} \cdot \mathbf{t}_{E_1} = -H_1$, with H_1 is the height of K_1 on E_1 . Now, by the fact $2|K| = |E_1| \cdot H_1$, we have

$$\nabla^\perp \lambda_2 \cdot \mathbf{n}_{E_1} = \nabla \lambda_2 \cdot \mathbf{t}_{E_1} = -1/|E_1|.$$

Similarly,

$$\nabla^\perp \lambda_3 \cdot \mathbf{n}_{E_1} = \nabla \lambda_3 \cdot \mathbf{t}_{E_1} = 1/|E_1|.$$

On the hand, since the tangential and normal vectors of an edge are orthogonal, we have

$$(4.1) \quad \nabla^\perp \lambda_2 \cdot \mathbf{n}_{E_2} = \nabla^\perp \lambda_3 \cdot \mathbf{n}_{E_3} = 0.$$

Globally, on $E_\ell = \{z_s, z_t\}$ ($s < t$), the unit tangential and normal vectors are

$$\mathbf{t}_{E_\ell} = \frac{1}{|E_\ell|} \begin{pmatrix} x_t - x_s \\ y_t - y_s \end{pmatrix} \quad \text{and} \quad \mathbf{n}_{E_\ell} = \frac{1}{|E_\ell|} \begin{pmatrix} y_t - y_s \\ x_s - x_t \end{pmatrix}.$$

We call the adjacent element whose out unit normal vector is the same as \mathbf{n}_{E_ℓ} as K_ℓ^- , and the element whose out unit normal vector is the opposite of \mathbf{n}_{E_ℓ} as K_ℓ^+ . On both K_ℓ^- and K_ℓ^+ , we have

$$(4.2) \quad \nabla^\perp \lambda_s \cdot \mathbf{n}_{E_\ell} = \nabla \lambda_s \cdot \mathbf{t}_{E_\ell} = -1/|E_\ell| \quad \text{and} \quad \nabla^\perp \lambda_t \cdot \mathbf{n}_{E_\ell} = \nabla \lambda_t \cdot \mathbf{t}_{E_\ell} = 1/|E_\ell|.$$

5. CONSTRUCTIONS OF EDGE-BASED BDM BASIS FUNCTIONS

Let λ_i be the standard linear Lagrange finite element basis function of the vertex z_i , i.e., it is piecewise linear on each element, globally continuous, 1 at z_i and 0 at other vertices.

For an edge $E_\ell = \{z_s, z_t\}$, $s < t$, $1 \leq \ell \leq \text{NE}$ with the globally fixed starting vertex s and terminal vertex t , its two BDM_1 basis functions associated with the edge are

$$(5.1) \quad \phi_{\ell,1} = \lambda_s \nabla^\perp \lambda_t \quad \text{and} \quad \phi_{\ell,2} = -\lambda_t \nabla^\perp \lambda_s.$$

It's clear that the basis functions are only non-zero in the two adjacent elements K_ℓ^- and K_ℓ^+ (one element in the case of boundary) of E_ℓ .

Lemma 5.1. *There hold*

$$(5.2) \quad \phi_{\ell,1} \cdot \mathbf{n}_{E_k}|_{E_k} = \begin{cases} 0, & \text{if } k \neq \ell; \\ \lambda_s/|E_\ell|, & \text{if } k = \ell; \end{cases} \quad \text{and} \quad \phi_{\ell,2} \cdot \mathbf{n}_{E_k}|_{E_k} = \begin{cases} 0, & \text{if } k \neq \ell; \\ \lambda_t/|E_\ell|, & \text{if } k = \ell, \end{cases}$$

and

$$(5.3) \quad \nabla \cdot \phi_{\ell,1} = \nabla \cdot \phi_{\ell,2} = \frac{1}{2|K^-|} \text{ on } K^- \quad \text{and} \quad \nabla \cdot \phi_{\ell,1} = \nabla \cdot \phi_{\ell,2} = -\frac{1}{2|K^+|} \text{ on } K^+.$$

Proof. By the discussion in Section 4.3, we have

$$\nabla^\perp \lambda_s \cdot \mathbf{n}_{E_\ell} = -1/|E_\ell| \quad \text{and} \quad \nabla^\perp \lambda_t \cdot \mathbf{n}_{E_\ell} = 1/|E_\ell|,$$

and

$$\phi_{\ell,1} \cdot \mathbf{n}_{E_\ell}|_{E_\ell} = \lambda_s/|E_\ell| \quad \text{and} \quad \phi_{\ell,2} \cdot \mathbf{n}_{E_\ell}|_{E_\ell} = \lambda_t/|E_\ell|.$$

Assume K_ℓ^- 's three counterclockwisely ordered vertices are $\{z_{r-}, z_s, z_t\}$. We call edge with two endpoints z_t and z_{r-} to be $E_{t,r-}$, and the edge with two endpoint z_{r-} and z_s

to be $E_{s,r-}$. Here the orientations of these two edges are not important. Then by (4.1), $\nabla^\perp \lambda_s \cdot \mathbf{n}_{E_{t,r-}} = 0$, and the fact λ_t is zero on $E_{s,r-}$, we get

$$\phi_{\ell,2} \cdot \mathbf{n}_E = 0, \text{ with } E = E_{s,r-} \text{ or } E_{t,r-}.$$

We can prove (5.2) is true for $\phi_{\ell,1}$ and K_ℓ^+ similarly.

The divergence part of the theorem can be easily proved by recalling the definition of λ_s and λ_t on K_ℓ^- and K_ℓ^+ , and notice that the counterclockwise ordering of vertices on K_ℓ^- is $\{z_{r-}, z_s, z_t\}$, and that on K_ℓ^+ is $\{z_{r+}, z_t, z_s\}$. \square

Remark 5.2. *There are other definitions of the BDM₁ basis functions. One of the popular choice is the following:*

$$(5.4) \quad \phi_{\ell,1} = \lambda_s \nabla^\perp \lambda_t - \lambda_t \nabla^\perp \lambda_s \quad \text{and} \quad \phi_{\ell,2} = \lambda_s \nabla^\perp \lambda_t + \lambda_t \nabla^\perp \lambda_s.$$

The first basis function coincides with the RT₀ basis function with property $\phi_{\ell,1} \cdot \mathbf{n}_{E_k} = |E_\ell| \delta_{\ell k}$. This set of choices of basis functions is hierarchical. The reason we choose (5.1) is that it is symmetric. If the reader needs the basis to be hierarchical, one should choose (5.4).

Remark 5.3. *The other versions of basis functions, for example, the RT basis function used in [2], choose $\phi_\ell^{rt} \cdot \mathbf{n}_{E_k} = \delta_{k\ell}$. The issue of this choice of basis functions is that the mass matrix has a condition number Ch_{\max}/h_{\min} , where h_{\max} and h_{\min} are the maximal and minimal diameters of the elements respectably. This will be a problem for an adaptively generated meth. Though it can be easily fixed by preconditioning it with the inverse of the diagonal matrix of it, we avoid it by choosing basis in (5.1) or (5.4).*

6. CONSTRUCTING AND SOLVING THE MATRIX PROBLEM

6.1. Assembling matrices.

6.1.1. Local matrices. For an element K , we want to compute the local contributions of this element to the matrices B and C .

We use `localEdge=[2 3; 3 1; 1 2]` to denote the local edge with respect to the local indices of vertices. For the i -th edge, we let `ii1 = localEdge(i,1)` and `ii2 = ... localEdge(i,2)` to denote the local starting and terminal vertices of it. When the local orientation and the global orientation of the edge E is different (`signedge` of the edge is -1), we need to switch the local order to find the right global starting and terminal vertices of edge. For a given i -th local edge, by the line `i1 = (signedge(:,i)>0).*ii1+ ... (signedge(:,i)<0).*ii2`, we get `i1=ii1` if the local orientation and the global orientation are the same, and `i1=ii2` otherwise. `i2` can be done similarly. Once we find the `i1` (local starting vertex) and `i2` (local terminal vertex) with right global orientation, we can get the corresponding coefficients a_{i1} and b_{i1} of $\lambda_{i1} = (a_{i1}x + b_{i1}y + c_{i1})/(2|K|)$, and the same things for `i2`. The function `BDMrightorder.m` does the above job. With an input of $i = 1, 2$, or 3 be the local vertex index of an element, this function returns the values `i1` and `i2`, which are the correct starting and terminal vertices of the corresponding edge with respect to the global fixed edge orientation, and `ai1, ai2, bi1, bi2` are the corresponding coefficients of λ_{i1} and λ_{i2} .

To compute the local contribution of an element K to B , the local mass matrix contains three cases, $(\alpha^{-1}\phi_{i,1}, \phi_{j,1})$, $(\alpha^{-1}\phi_{i,1}, \phi_{j,2})$, and $(\alpha^{-1}\phi_{i,2}, \phi_{j,2})$ with

$$\phi_{i,1} = \lambda_{i1} \nabla^\perp \lambda_{i2}, \quad \phi_{i,2} = -\lambda_{i2} \nabla^\perp \lambda_{i1}, \quad \phi_{j,1} = \lambda_{j1} \nabla^\perp \lambda_{j2}, \quad \text{and} \quad \phi_{j,2} = -\lambda_{j2} \nabla^\perp \lambda_{j1}.$$

LISTING 5. BDMrightorder.m

```

1 function [i1,i2,ai1,bil,ai2,bi2] = BDMrightorder(i,signedge,NT,a,b)
2 localEdge = [2 3; 3 1; 1 2];
3 i1 = localEdge(i,1); i2 = localEdge(i,2);
4 i1 = (signedge(:,i)>0).*i1+ (signedge(:,i)<0).*i2;
5 i2 = (signedge(:,i)<0).*i1+ (signedge(:,i)>0).*i2;
6 ai1 = a((i1-1)*NT+(1:NT)'); ai2 = a((i2-1)*NT+(1:NT)');
7 bil = b((i1-1)*NT+(1:NT)'); bi2 = b((i2-1)*NT+(1:NT)');
8 end

```

For barycentric coordinates, $\int_K \lambda_i \lambda_j dx = (1 + \delta_{i,j})|K|/12$. An easy computation shows that

$$\begin{aligned} (\alpha^{-1}\phi_{i,1}, \phi_{j,1})_K &= \alpha^{-1}(1 + \delta_{i1,j1})(a_{i2} \cdot a_{j2} + b_{i2} \cdot b_{j2})/(48|K|); \\ (\alpha^{-1}\phi_{i,1}, \phi_{j,2})_K &= -\alpha^{-1}(1 + \delta_{i1,j2})(a_{i2} \cdot a_{j1} + b_{i2} \cdot b_{j1})/(48|K|); \\ (\alpha^{-1}\phi_{i,2}, \phi_{j,2})_K &= \alpha^{-1}(1 + \delta_{i2,j2})(a_{i1} \cdot a_{j1} + b_{i1} \cdot b_{j1})/(48|K|). \end{aligned}$$

For $k = 1, 2$, $\nabla \cdot \phi_{i,k} = 1/(2|K|)$ when the i -th edge has the same orientation as the global edge, and $-1/(2|K|)$ otherwise. Thus, denote $s(i)$ be the value of `signedge(:, i)`, we have

$$-(\nabla \cdot \phi_{i,1}, 1)_K = -(\nabla \cdot \phi_{i,2}, 1)_K = -s(i)/2.$$

The local matrix (here we abuse the notations to use local $\phi_{i,k}$, $i = 1 \dots 3$, $k = 1, 2$ to denote the local BDM basis functions):

$$-((\nabla \phi_{1,1}, 1)_K, (\nabla \phi_{2,1}, 1)_K, (\nabla \phi_{3,1}, 1)_K, (\nabla \phi_{1,2}, 1)_K, (\nabla \phi_{2,2}, 1)_K, (\nabla \phi_{3,2}, 1)_K)$$

is simply $-(s(1), s(2), s(3), s(1), s(2), s(3))/2$, or `-[signedge(:, 1:3), signedge(:, 1:3)]` in MATLAB code.

6.1.2. Assembling global matrices. For a local index i , its corresponding global edge index is `double(elem2edge(:, i))`. MATLAB function `sparse` is used to generate the global matrices from local contributions. This is one of the key step to ensure the vectorization of the MATLAB finite element code, see [10, 11] for more detailed discussions on `sparse`.

Here are some comments of the MATLAB code `assemblebdm.m`.

- Line 1: The function is called by


```
A = assemblebdm(NT, NE, a, b, area, elem2edge, signedge, inva)
```

 where A is a $(2*NE+NT) \times (2*NE+NT)$ matrix, $a, b, area$ are the coefficients of local barycentric coordinates, `signedge` is the $NT \times 3$ matrix about the local and global edge orientations, and `inva` is $NT \times 1$ vector of α^{-1} .
- Lines 4-15: We generates the matrix $B = (\alpha^{-1}\sigma_h, \tau_h)$, σ_h and τ_h in BDM_1 by assembling local element-wise contributions.
- Lines 6-7: We generates the right starting and terminal indices of a global edge in the local element, and their corresponding coefficients of local barycentric coordinates.
- Lines 8-10: We compute E , H , and G , which are $(\alpha^{-1}\phi_{i,1}, \phi_{j,1})_K$, $(\alpha^{-1}\phi_{i,1}, \phi_{j,2})_K$, and $(\alpha^{-1}\phi_{i,2}, \phi_{j,2})_K$, respectively.
- Lines 11-13: B is generated by `sparse`.

LISTING 6. assemblebdm.m

```

1 function A = assemblebdm(NT,NE,a,b,area,elem2edge,signedge,inva)
2 B = sparse(2*NE, 2*NE); C = sparse(NT, 2*NE); D = sparse(NT,NT);
3 %% Blocal(6,6) = [E(3,3) H(3,3); H'(3,3) G(3,3)]
4 for i = 1:3
5     for j = 1:3
6         [i1,i2,ai1,bi1,ai2,bi2] = BDMrightorder(i,signedge,NT,a,b);
7         [j1,j2,aj1,bj1,aj2,bj2] = BDMrightorder(j,signedge,NT,a,b);
8         E = inva./(48*area).* (1+(i1==j1)).*(ai2.*aj2+bi2.*bj2);
9         H = -inva./(48*area).* (1+(i1==j2)).*(ai2.*aj1+bi2.*bj1);
10        G = inva./(48*area).* (1+(i2==j2)).*(ai1.*aj1+bi1.*bj1);
11        ii = double(elem2edge(:,i)); jj = double(elem2edge(:,j));
12        B = B + sparse(ii,jj,E,2*NE,2*NE)+sparse(ii,jj+NE,H,2*NE,2*NE) ...
13            +sparse(jj+NE,ii,H,2*NE,2*NE)+sparse(NE+ii,NE+jj,G,2*NE,2*NE);
14    end
15 end
16 for i = 1:3
17     ii = double(elem2edge(:,i));
18     C = C + sparse(1:NT,ii,-signedge(:,i)/2,NT,2*NE) ...
19         +sparse(1:NT,ii+NE,-signedge(:,i)/2,NT,2*NE);
20 end
21 A = [B C';C D];
22 end

```

- Lines 16-20: C is generated by local contributions $[-\text{signedge}(:,1:3), \dots, -\text{signedge}(:,1:3)]$.
- Lines 21: A is generated by adding a zero matrix D on 22 block.

6.2. Assembling the force f term. Since $\nabla \cdot \boldsymbol{\sigma}_h \in P_0$, we only need the numerical integration of the f term to be accurate as if f is a constant on each element. Thus, the term related to $-(f, 1)_K$ can be computed by a one-point quadrature rule:

$$-\int_K f dx \approx -f(x_{mid}, y_{mid})|K|,$$

where (x_{mid}, y_{mid}) are the coordinates of the gravity center of the element K .

6.3. Generating boundary data. On the boundary, we need `sign_D` and `sign_N` to denote the difference between orientations inherited from the local edge ordering of the element and the global edge orientations like `edgesign`. Since the unit out normal vector of an element on the boundary is the same as the unit out normal vector of the whole domain, `sign_D` and `sign_N` are actually the difference of normal directions of Dirichlet and Neumann edges and global out normal directions.

The MATLAB function `boundary.m` generates the Dirichlet and Neumann edges and their orientations with respect to the global edges.

- Lines 3-10: We generates un-sorted Dirichlet and Neumann edges and their signs.
- Lines 11-13: We generates sorted Dirichlet and Neumann edges and their signs.

We denote the set of indices of edges on the Dirichlet and Neumann boundary to be `ind_D` and `ind_N`, respectively.

LISTING 7. boundary.m

```

1 function [Dirichlet, Neumann, sign_D, sign_N] = boundary(elem,bdEdge)
2 NT = size(elem,1);
3 totalEdge = [elem(:,[2,3]); elem(:,[3,1]); elem(:,[1,2])];
4 isBdEdge = reshape(bdEdge,3*NT,1);
5 Dirichlet = totalEdge((isBdEdge == 1),:);
6 Neumann = totalEdge((isBdEdge == 2),:);
7 NE_D = size(Dirichlet,1); NE_N = size(Neumann,1);
8 sign_D = ones(NE_D,1); sign_N = ones(NE_N,1);
9 sign_D = sign_D(:,1) - 2* (Dirichlet(:,1)>Dirichlet(:,2));
10 sign_N = sign_N(:,1) - 2* (Neumann(:,1)>Neumann(:,2));
11 Dirichlet = sort(Dirichlet,2); Neumann = sort(Neumann,2);
12 [Dirichlet,I_d] = sortrows(Dirichlet); [Neumann,I_n] = sortrows(Neumann);
13 sign_D = sign_D(I_d); sign_N = sign_N(I_n);
14 end

```

To computer the term $-(\boldsymbol{\tau} \cdot \mathbf{n}, g_D)_{\Gamma_D}$, we need to be careful about two things. One is the difference between unit out normal vector of the domain and that of the edge, the other one is the numerical quadrature formula. In order to guarantee the convergence order of BDM_1 element, we should use two-point numerical quadrature on an edge. The 2-point Gauss-Legendre quadrature of a function $f(x)$ on interval $[a, b]$ is:

$$(6.1) \quad \int_a^b f(x)dx \approx \frac{b-a}{2} \left(f\left(\frac{a+b}{2} - \frac{b-a}{2\sqrt{3}}\right) + f\left(\frac{a+b}{2} + \frac{b-a}{2\sqrt{3}}\right) \right).$$

Note that on a Dirichlet edge $E_j = \{z_s, z_t\}$, $s < t$, $\phi_{j,1} \cdot \mathbf{n}_{E_j} = \lambda_s/|E_j|$ and $\phi_{j,2} \cdot \mathbf{n}_{E_j} = \lambda_t/|E_j|$. Thus, to compute $-\int_{E_j} (\phi_{j,\ell} \cdot \mathbf{n}_\Omega) g_D dx$ by formula (6.1), we need the value of g_D at quadrature points $p_1 = \frac{n_1+n_2}{2} - \frac{n_2-n_1}{2\sqrt{3}}$ and $p_2 = \frac{n_1+n_2}{2} + \frac{n_2-n_1}{2\sqrt{3}}$, where n_1 and n_2 are the coordinates of the z_s and z_t , respectively. We also need to know the value of λ_s and λ_t at p_1 and p_2 , which are

$$\lambda_s(p_1) = \lambda_t(p_2) = \frac{1}{2} + \frac{1}{2\sqrt{3}} \quad \text{and} \quad \lambda_s(p_2) = \lambda_t(p_1) = \frac{1}{2} - \frac{1}{2\sqrt{3}}.$$

Thus, by letting $s(j) = \text{sign_D}(j)$, $s(j) = 1$ or -1 be the number denoting the difference between the local and global edge orientation on Γ_D , we have

$$\begin{aligned} - \int_{E_j} (\phi_{j,1} \cdot \mathbf{n}_\Omega) g_D dx &\approx -s(j) \left(g_D(p_1)\left(\frac{1}{4} + \frac{1}{4\sqrt{3}}\right) + g_D(p_2)\left(\frac{1}{4} - \frac{1}{4\sqrt{3}}\right) \right), \\ - \int_{E_j} (\phi_{j,2} \cdot \mathbf{n}_\Omega) g_D dx &\approx -s(j) \left(g_D(p_1)\left(\frac{1}{4} - \frac{1}{4\sqrt{3}}\right) + g_D(p_2)\left(\frac{1}{4} + \frac{1}{4\sqrt{3}}\right) \right). \end{aligned}$$

To handle the Neumann boundary condition, on each $E_j = \{z_s, z_t\} \in \mathcal{E}_N$, we want to compute $\boldsymbol{\sigma}_N|_{E_j} = c_{j,1}\phi_{j,1} + c_{j,2}\phi_{j,2}$. Then $\boldsymbol{\sigma}_N \cdot \mathbf{n}_\Omega|_{E_j} = c_{j,1}\phi_{j,1} \cdot \mathbf{n}_\Omega + c_{j,2}\phi_{j,2} \cdot \mathbf{n}_\Omega$. Let $s(j) = 1$ or -1 be the number denoting the difference between the local and global edge orientation on E_j , we should have

$$s(j)(c_{j,1}\lambda_s + c_{j,2}\lambda_t)/|E_j| \approx g_N.$$

LISTING 8. rhsdide.m

```

1 function [A,b,sol,freeDof]=rhsdide(node,elem,edge,bdEdge,area,A,sol,f,gD,gN)
2 NT = size(elem,1); NE = size(edge,1);
3 [Dirichlet, Neumann,sign_D,sign_N] = boundary(elem, bdEdge);
4 %% Assemble right hand side.
5 mid = (node(elem(:,1),:)+node(elem(:,2),:)+node(elem(:,3),:))/3;
6 b2 = accumarray((1:NT)', -f(mid).*area, [NT 1]);
7 %% Dirichelet BC
8 n1= node(Dirichlet(:,1),:); n2= node(Dirichlet(:,2),:);
9 p1=(n1-n2)/2*sqrt(1/3)+(n2+n1)/2; p2=(n2-n1)/2*sqrt(1/3)+(n2+n1)/2;
10 intgDphi1n = (gD(p1)*(1+sqrt(1/3))+gD(p2)*(1-sqrt(1/3)))/4;
11 intgDphi2n = (gD(p2)*(1+sqrt(1/3))+gD(p1)*(1-sqrt(1/3)))/4;
12 [useless, ind_D] = intersect(edge, Dirichlet, 'rows');
13 bb1 = accumarray(ind_D,-sign_D.*intgDphi1n,[NE 1]);
14 bb2 = accumarray(ind_D,-sign_D.*intgDphi2n,[NE 1]);
15 b1 = [bb1;bb2];
16 %% Neumann BC
17 n1= node(Neumann(:,1),:); n2= node(Neumann(:,2),:);
18 p1=(n1-n2)/2*sqrt(1/3)+(n2+n1)/2; p2=(n2-n1)/2*sqrt(1/3)+(n2+n1)/2;
19 edgeLength_N = sqrt(sum((n1-n2).^2,2));
20 intgNlams = edgeLength_N.* (gN(p1)*(1+sqrt(1/3))+gN(p2)*(1-sqrt(1/3)))/4;
21 intgNlamt = edgeLength_N.* (gN(p2)*(1+sqrt(1/3))+gN(p1)*(1-sqrt(1/3)))/4;
22 [useless, ind_N] = intersect(edge, Neumann, 'rows');
23 sol(ind_N) = 2*sign_N.* (2*intgNlams-intgNlamt);
24 sol(ind_N+NE) = 2*sign_N.* (2*intgNlamt-intgNlams);
25 %% modify right hand side
26 b = [b1;b2]; b = b - A*sol;
27 %%freeDof
28 isBdDof = false(2*NE+NT,1); isBdDof(ind_N)=true; isBdDof(ind_N+NE)=true;
29 freeDof = find(~isBdDof);
30 end

```

Let the L^2 -projection of g_N on to $P_1(E_j) = \text{span}\{\lambda_s, \lambda_t\}$ to be $g_{N,h} = d_s \lambda_s + d_t \lambda_t$:

$$\begin{pmatrix} (\lambda_s, \lambda_s)_{E_j} & (\lambda_t, \lambda_s)_{E_j} \\ (\lambda_s, \lambda_t)_{E_j} & (\lambda_t, \lambda_t)_{E_j} \end{pmatrix} \begin{pmatrix} d_s \\ d_t \end{pmatrix} = \begin{pmatrix} (g_N, \lambda_s)_{E_j} \\ (g_N, \lambda_t)_{E_j} \end{pmatrix}.$$

Since $\int_E \lambda_i \lambda_j ds = |E|(1 + \delta_{ij})/6$, replace $(g_N, \lambda_s)_{E_j}$ and $(g_N, \lambda_t)_{E_j}$, by $I_{j,s}$ and $I_{j,t}$ using the two-point numerical quadrature (6.1) as we did for g_D , we get

$$d_s = (4I_{j,s} - I_{2j,t})/|E_j| \quad \text{and} \quad d_t = (4I_{j,t} - 2I_{j,s})/|E_j|.$$

So $\sigma_N|_{E_j} = c_{j,1}\phi_{j,1} + c_{j,2}\phi_{j,2}$ with $c_{j,1} = s(j)(4I_{j,s} - 2I_{j,t})$ and $c_{j,2} = s(j)(4I_{j,t} - 2I_{j,s})$, or, in the notation of (2.9),

$$x_j = s(j)(4I_{j,s} - 2I_{j,t}) \quad \text{and} \quad x_{NE+j} = s(j)(4I_{j,t} - 2I_{j,s}).$$

With this know σ_N , then the right-hand side of the discrete problem can be easily handled as in [1, 10].

The followings are some comments about the function `rhsdide.m` of handling the f term and boundary conditions.

- Lines 5-6: We generate terms related to $-(f, v)$.

LISTING 9. diffusionbdm.m

```

1 function [sigma,u] = diffusionbdm(node,elem,bdEdge,elem2edge,edge, ...
2     signededge,exactalpha,f,gD,gN)
3 NT = size(elem,1); NE = size(edge,1);
4 sol = zeros(2*NE+NT,1);
5 inva = 1./exactalpha((node(elem(:,1))+node(elem(:,2))+node(elem(:,3)))/3);
6 [a,b,area] = gradlambda(node,elem);
7 A = assemblebdm(NT,NE,a,b,area,elem2edge,signededge,inva);
8 [A,b,sol,freeDof] = rhsde(node,elem,edge,bdEdge,area,A,sol,f,gD,gN);
9 sol(freeDof) = A(freeDof,freeDof)\b(freeDof);
10 sigma = sol(1:2*NE); u = sol(2*NE+1:end);
11 end

```

- Lines 8-15: We generate terms related to $-(\boldsymbol{\tau} \cdot \mathbf{n}, g_D)_{\Gamma_D}$.
- Lines 17-24: We construct the $\boldsymbol{\sigma}_N$.
- Line 26: We handle the right-hand side of the matrix problem.
- Lines 28-29: We find the degrees of freedom of the matrix problem. (Those terms related to $\boldsymbol{\sigma}_N$ are known, thus not free.)

6.4. Solving the BDM mixed problem. The MATLAB function `diffusionbdm.m` is the principle function to solve the BDM mixed problem. With all the building blocks given before, it is relatively easy. Line 5 is used to got α^{-1} in each element. All the rest lines are self-explanatory.

7. CHECKING ERRORS

The final step of a numerical test is often the convergence test by computing some norms of the error between the exact and numerical solutions obtained for different mesh sizes. In our case, we need to compute

$$\|\alpha^{-1/2}(\boldsymbol{\sigma} - \boldsymbol{\sigma}_h)\|_0 \quad \text{and} \quad \|u - u_h\|_0.$$

For the both terms $\|\alpha^{-1/2}(\boldsymbol{\sigma} - \boldsymbol{\sigma}_h)\|_0$ and $\|u - u_h\|_0$, the direct way to compute them is summing up the errors on each element by direct computations. To lower the numerical quadrature order, the first step should be

$$\begin{aligned} \|\alpha^{-1/2}(\boldsymbol{\sigma} - \boldsymbol{\sigma}_h)\|_0^2 &= (\alpha^{-1}(\boldsymbol{\sigma} - \boldsymbol{\sigma}_h), \boldsymbol{\sigma} - \boldsymbol{\sigma}_h) = (\alpha^{-1}\boldsymbol{\sigma}, \boldsymbol{\sigma}) - 2(\alpha^{-1}\boldsymbol{\sigma}, \boldsymbol{\sigma}_h) + (\alpha^{-1}\boldsymbol{\sigma}_h, \boldsymbol{\sigma}_h), \\ \|u - u_h\|_0^2 &= (u - u_h, u - u_h) = (u, u) - 2(u, u_h) + (u_h, u_h). \end{aligned}$$

The terms $(\alpha^{-1}\boldsymbol{\sigma}, \boldsymbol{\sigma})$ and (u, u) can be computed exactly by softwares like Mathematica and Maple. In our example, $(\alpha^{-1}\boldsymbol{\sigma}, \boldsymbol{\sigma}) = 1993/75 \approx 26.5733$ and $(u, u) = 18131/7500 \approx 2.41747$. The terms $(\alpha^{-1}\boldsymbol{\sigma}_h, \boldsymbol{\sigma}_h)$ and (u_h, u_h) can also be computed exactly. For the terms $(\alpha^{-1}\boldsymbol{\sigma}, \boldsymbol{\sigma}_h)$ and (u_h, u_h) , numerical quadratures must be used.

Since this part of codes is less important, we only give brief comments about the functions we used. The MATLAB function `exactsigma.m` returns the exact value of $\boldsymbol{\sigma}$. The function `sigmahBDM.m` returns the values of $\boldsymbol{\sigma}_h$ at those numerical quadrature points. The function `errorBDM.m` is used to compute the errors. Here, we use a 6-point quadrature to compute them. Matrix `xw` is a 6×3 matrix, with `xw(:,1:2)` are the coordinates such that the quadrature point is $p=p1*(1-xw(i,1)-xw(i,2))+p2*xw(i,1)+p3*xw(i,2)$, where

LISTING 10. exactsigma.m

```

1 function [sigma1,sigma2] = exactsigma(p)
2 x = p(:,1); y = p(:,2);
3 sigma1=-2*x.*y.*y-1;
4 sigma2=(x<0).*(-2*x.*x.*y-10)+(x>=0).*(-2*x.*x.*y-1);
5 end

```

LISTING 11. sigmahBDM.m

```

1 function [sigmah1,sigmah2] = ...
2     sigmahBDM(elem,node,NE,elem2edge,signedge,w1,w2,sigma)
3 [a,b,area] = gradlambda(node,elem); NT = size(elem,1);
4 lambda_at_w = [1-w1-w2,w1,w2];
5 sigmah1 = zeros(NT,1);           sigmah2 = zeros(NT,1);
6 for i = 1:3
7     [i1,i2,ai1,bi1,ai2,bi2] = BDMrightorder(i,signedge,NT,a,b);
8     ii = double(elem2edge(:,i));
9     sigmah1 = sigmah1+sigma(ii).*lambda_at_w(i1)'.*bi2./area/2 ...
10        -sigma(ii+NE).*lambda_at_w(i2)'.*bi1./area/2;
11     sigmah2 = sigmah2-sigma(ii).*lambda_at_w(i1)'.*ai2./area/2 ...
12        +sigma(ii+NE).*lambda_at_w(i2)'.*ai1./area/2;
13 end
14 end

```

p_1 , p_2 , and p_3 are the coordinates of three vertices; $xw(:, 3)$ is the corresponding weight for the point.

We use the code listed in Listing 13 to compute the error.

The original mesh given has mesh size $h = 1$. We refine the mesh uniformly several times, for example, using $[node, elem, bdEdge] = \text{uniformbisection}(node, elem, bdEdge)$, where `uniformbisection` is a uniform refinement MATLAB function given in the package *iFEM* [10]. We compute the errors for different mesh sizes, and have the Table 1. We get

$$\frac{\|\alpha^{-1}(\boldsymbol{\sigma} - \boldsymbol{\sigma}_h)\|_0}{\|\alpha^{-1}(\boldsymbol{\sigma} - \boldsymbol{\sigma}_{h/2})\|_0} \approx 4 \quad \text{and} \quad \frac{\|u - u_h\|_0}{\|u - u_{h/2}\|_0} \approx 2.$$

This result is in agreement with the a priori error estimates (2.6).

8. RELATED FINITE ELEMENTS

Once we know how to program BDM_1 methods, we can actually write codes for similar finite elements, with two things in mind: Writing basis functions in barycentric coordinates and correcting the local edge orientation. For more higher order elements, we may use Legendre polynomials instead of Lagrange polynomials, but the idea is similar.

8.1. Other $H(\text{div})$ -conforming edge-based basis in 2D. For RT_0 element, as mentioned in Remark 5.2, its basis function on an edge can be written as

$$\phi_\ell^{rt} = \lambda_s \nabla^\perp \lambda_t - \lambda_t \nabla^\perp \lambda_s$$

LISTING 12. errorBDM.m

```

1 function [error_sigma, error_u] = errorBDM(node, elem, NE, area, elem2edge, ...
2     signedge, inva, sigma, u, ss, uu)
3 n1 = elem(:,1); n2 = elem(:,2); n3 = elem(:,3);
4 p1 = node(n1,:); p2 = node(n2,:); p3 = node(n3,:);
5 xw=[0.44594849091597 0.44594849091597 0.22338158967801;...
6     0.44594849091597 0.10810301816807 0.22338158967801;...
7     0.10810301816807 0.44594849091597 0.22338158967801;...
8     0.09157621350977 0.09157621350977 0.10995174365532;...
9     0.09157621350977 0.81684757298046 0.10995174365532;...
10    0.81684757298046 0.09157621350977 0.10995174365532];
11 NP =size(xw,1); NT = size(elem,1);
12 uuhlocal = zeros(NT,1); % (u, u_h)_K
13 sshlocal = zeros(NT,1); % (\alpha^{-1}\sigma, \sigma_h)_K
14 shshlocal = zeros(NT,1); % (\alpha^{-1}\sigma_h, \sigma_h)_K
15 for i=1:NP
16     p=p1*(1-xw(i,1)-xw(i,2))+p2*xw(i,1)+p3*xw(i,2);
17     uuhlocal = uuhlocal + exactu(p)*xw(i,3);
18     [sigma1,sigma2] = exactsigma(p);
19     [sigmah1,sigmah2] = ...
20         sigmahBDM(elem, node, NE, elem2edge, signedge, xw(i,1), xw(i,2), sigma);
21     sshlocal = sshlocal + (sigma1.*sigmah1+sigma2.*sigmah2)*xw(i,3);
22     shshlocal = shshlocal + (sigmah1.*sigmah1+sigmah2.*sigmah2)*xw(i,3);
23 end
24 uuhlocal = u.*uuhlocal.*area;           uhuhlocal = u.*u.*area;
25 sshlocal = inva.*sshlocal.*area;        shshlocal = inva.*shshlocal.*area;
26 error_u = sqrt(abs(uu-2*sum(uuhlocal)+sum(uhuhlocal)));
27 error_sigma = sqrt(abs(ss-2*sum(sshlocal)+sum(shshlocal)));
28 end

```

LISTING 13. checking errors

```

1 uu=18131/7500; ss=1993/75;
2 [error_sigma, error_u] = errorBDM(node, elem, NE, area, elem2edge, ...
3     signedge, inva, sigma, u, ss, uu);

```

TABLE 1. Errors of σ and u on different mesh sizes

h	$\ \alpha^{-1}(\sigma - \sigma_h)\ _0$	$\ \alpha^{-1}(\sigma - \sigma_{h/2})\ _0$	$\ u - u_h\ _0$	$\frac{\ u - u_h\ _0}{\ u - u_{h/2}\ _0}$
1	1.6968e-01		4.9712e-01	
1/2	4.2091e-02	4.0314	2.4400e-01	2.0374
1/4	1.0600e-02	3.9707	1.2118e-01	2.0135
1/8	2.6630e-03	3.9805	6.0481e-02	2.0037
1/16	6.6739e-04	3.9901	3.0226e-03	2.0009
1/32	1.6705e-04	3.9952	1.5111e-03	2.0002
1/64	4.1788e-05	3.9975	7.5555e-04	2.0001

Actually, it is insensitive to the starting and terminal vertices:

$$\phi_\ell^{rt} = \lambda_s \nabla^\perp \lambda_t - \lambda_t \nabla^\perp \lambda_s = \lambda_t \nabla^\perp \lambda_s - \lambda_s \nabla^\perp \lambda_t,$$

and on adjacent elements $K^- = \{z_{r-}, z_s, z_t\}$ and $K^+ = \{z_{r+}, z_t, z_s\}$,

$$\phi_\ell^{rt}|_{K^-} = \frac{1}{2|K^-|} \begin{pmatrix} x - x_{r-} \\ y - y_{r-} \end{pmatrix} \quad \text{and} \quad \phi_\ell^{rt}|_{K^+} = -\frac{1}{2|K^+|} \begin{pmatrix} x - x_{r+} \\ y - y_{r+} \end{pmatrix}.$$

Thus lines 4-5 of `BDMrightorder` is unnecessary for RT_0 elements.

For all RT_k and BDM_k spaces, basis functions are divided into two categories, edge based functions and element-based functions. The element based functions are usually easy to construct since they are only non-zero in one element, see [4]. So we will only discuss edge based basis functions. Both RT_k and BDM_k spaces have $k+1$ basis functions on each edge. We only need to functions to span $P_k(E_\ell)$ on E_ℓ . For BDM_2 and RT_2 , three edge basis functions on E_ℓ are

$$\phi_{\ell,1}^1 = \lambda_s^2 \nabla^\perp \lambda_t, \quad \phi_{\ell,2}^1 = \lambda_s \lambda_t (\nabla^\perp \lambda_t - \nabla^\perp \lambda_s), \quad \text{and} \quad \phi_{\ell,3}^1 = -\lambda_t^2 \nabla^\perp \lambda_s.$$

That is, we use λ_s^2 , λ_t^2 , and $\lambda_s \lambda_t$ to span P_2 on E_ℓ . We can use other choices, for example, Legendre polynomials for high order spaces. With this observation, the code developed in this paper can be easily adapted to these spaces.

8.2. Nédélec spaces in 2D. For Nédélec spaces, the 1st and 2nd types Nédélec spaces correspond to their $H(\text{div})$ counterparts are RT and BDM spaces, respectively. We also only need to discuss the construction of basis functions on edges. By (4.2), the zero-moment $H(\text{curl})$ edge basis function is

$$\psi_\ell^{ned} = \lambda_s \nabla \lambda_t - \lambda_t \nabla \lambda_s.$$

The linear $H(\text{curl})$ edge basis functions are

$$\psi_{\ell,1}^{ned} = \lambda_s \nabla \lambda_t \quad \text{and} \quad \psi_{\ell,2}^{ned} = -\lambda_t \nabla \lambda_s.$$

8.3. $H(\text{div})$ and $H(\text{curl})$ and basis functions in 3D. For a tetrahedral mesh, $H(\text{div})$ basis functions are defined on faces. Like the edge structure in this paper, we should define a face matrix. If $F_\ell = \{z_r, z_s, z_t\}$, we need ensure $r < s < t$, and choose the normal direction such that the area of $\{z_r, z_s, z_t\}$ is positive. Once this is done, we can do similar things as in 2D. The BDM_1 basis functions in 3D on a face $F_\ell = \{z_r, z_s, z_t\}$ are

$$\lambda_r \nabla \lambda_s \times \nabla \lambda_t, \quad \lambda_s \nabla \lambda_t \times \nabla \lambda_r, \quad \text{and} \quad \lambda_t \nabla \lambda_r \times \nabla \lambda_s.$$

Its RT_0 basis function on F is

$$\lambda_r \nabla \lambda_s \times \nabla \lambda_t + \lambda_s \nabla \lambda_t \times \nabla \lambda_r + \lambda_t \nabla \lambda_r \times \nabla \lambda_s.$$

For three dimensionalal $H(\text{curl})$ space, basis functions in barycentric coordinates can be found in [12].

REFERENCES

- [1] J. ALBERTY, C. CARSTENSEN, AND S.A. FUNKEN, *Remarks around 50 lines of Matlab: short finite element implementation*, Numer. Algoritm. 20, 117-137 (1999) 1, 6.3
- [2] C. BAHRIAWATI AND C. CARSTENSEN, *Three Matlab implementations of the lowest-order Raviart-Thomas mfem with a posteriori error control*, Comput. Methods Appl. Math., Vol.5(2005), No.4, pp.333-361. 1, 5.3

- [3] F. BREZZI; J. DOUGLAS, AND L. D MARINI, *Two families of mixed finite elements for second order elliptic problems*, Numer. Math., vol 47. no. 2., 1985. 217–235. 1, 2.2
- [4] D. BOFFI, F. BREZZI, AND M. FORTIN, *Mixed Finite Element Methods and Applications*, Springer Series in Computational Mathematics, 44, Springer, 2013. 1, 2.1, 8.1
- [5] Z. CAI AND S. ZHANG, *Recovery-based error estimator for interface problems: conforming linear elements*, SIAM J. Numer. Anal., Vol. 47, No. 3, pp. 2132–2156, 2009. 1
- [6] Z. CAI AND S. ZHANG, *Recovery-based error estimator for interface problems: mixed and nonconforming elements*, SIAM J. Numer. Anal, Vol. 48, No. 1, pp. 30–52, 2010. 1
- [7] Z. CAI AND S. ZHANG, *Flux recovery and a posteriori error estimators: Conforming elements for scalar elliptic equations*, SIAM J. Numer. Anal., Vol. 48, No. 2, pp. 578–602, 2010. 1
- [8] Z. CAI AND S. ZHANG, *Robust equilibrated residual error estimator for diffusion problems: Conforming elements*, SIAM J. Numer. Anal., Vol. 50, No. 1, pp. 151–170, 2012. 1
- [9] Z. CAI AND S. ZHANG, *Improved ZZ a posteriori error estimators for diffusion problems: conforming linear elements*, arXiv Preprint, arXiv:1508.00191. 1
- [10] L. CHEN. *iFEM: an integrated finite element methods package in MATLAB*, Technical Report, University of California at Irvine. 2009. 1, 4.1, 4.1, 6.1.2, 6.3, 7
- [11] S. FUNKEN, D. PRAETORIUS, AND P. WISSGOTT, *Efficient implementation of adaptive P1-FEM in MATLAB*, Comput. Methods Appl. Math. 11, 460-490 (2011) 1, 6.1.2
- [12] J. GOPALAKRISHNAN, L. F. DEMKOWICZ AND L. E. GARCA-CASTILLO, *Nédélec spaces in affine coordinates*, Computers and Mathematics with Applications, Volume 49, Issue 7-8, pp. 1285-1294, 2005. 8.3
- [13] P. A. RAVIART AND I. M. THOMAS, *A mixed finite element method for second order elliptic problems*, Lect. Notes Math. 606, Springer-Verlag, Berlin and New York (1977), 292-315. 1
- [14] P. SOLIN, K. SEGETH, AND I. DOLEZEL, *Higher-Order Finite Element Methods*, CRC Press, 2003. 1
- [15] L. N. TREFETHEN, *Spectral Methods in MATLAB*, SIAM, Philadelphia, 2000. 1

DEPARTMENT OF MATHEMATICS, CITY UNIVERSITY OF HONG KONG, KOWLOON TONG, HONG KONG SAR, CHINA

E-mail address: shun.zhang@cityu.edu.hk

URL: <http://personal.cityu.edu.hk/~szhang26>